Software Testing Report

Team 13

Darcy Adams

Davids Kacs

Jakub Grzmil

Nam Duong

Samin Alborzi Mohavved

Will Dunlop

Testing Documentation:https://team-13-rlc.github.io/pdfs/TestDoc2.pdf
Traceability Matrix:https://team-13-rlc.github.io/pdfs/TraceMat2.pdf
a)

For our codebase we used a mixture of black and white box testing. For the most part we found that our codebase was far more suited to white box testing and therefore we produced far more tests following white box methodologies. You can see this quite clearly in the testing documentation document that is linked at the top of this document. The reason we found white box testing more suitable is because our code is relatively modular and when it becomes non-modular we had to perform manual testing due to the limitations of testing libGDX ui.

For the majority of our automated tests we used Junit4 with mockito. The reason we chose to use Junit4 was because in our research we were not able to find much material online about utilising Junit5 with regards to LibGDX. This caused substantial issues when we tried to set up a Junit5 framework as we discovered that the changes from Junit4 to Junit5 were substantial and due to the relatively low experience level with Junit and testing in general we decided it would be simpler to use the more documented version of Junit. Furthermore we found a test runner[1] for LibGDX which was compatible with Junit4 only and made setting up our testing framework much simpler. This test runner was available under the apache licence and is properly marked within our codebase. We used mockito to allow some of our more complex code to be tested without being dependent on other parts of the codebase (which may or may not have been tested). The majority of our tests are not reliant on mockito however.

Some of our tests were manual tests, the reason why we couldn’t automate these tests is because they generally relate to user interface for which there is no testing framework in LibGDX. Or they relate to user experience which cannot easily be quantified.

Due to the time limited nature of this project we focused our testing on the items that we were adding to the project and the classes that those items were dependent upon.

b)

The testing table for this project can be found at the URL that is mentioned at the top of this document marked as Testing Documentation. This document contains most of the relevant technical information relating to testing within the project. As such in this document I will mainly talk about the rationale of certain tests as opposed to restating the data available there.

We split up our testing based upon which general system the tests relate to. For example all the tests that are related directly to the Collidable class have their id's starting with CollidableTest. The exception to this rule is for manual testing which we have classified as its own separate category due to how different in nature the tests are.

Before discussing specific tests I would like to state that in our current implementation all of our tests pass. However due to the time constraints of the project and the relative inexperience with testing within our group I wouldn't call our tests massively complete. This is partially shown by the fact that we only achieved 37% line coverage with our tests and 55% class coverage. As well as time, one of the main reasons that our coverage is so low is because we found the LibGDX screens very hard to test automatically and as such that dramatically reduced our coverage. As for correctness, for the most part I would say our tests are correct in that they accurately and usefully test our code base. This can be evidenced by the fact that we were able to use our tests to discover and then resolve bugs during implementation.

The tests ToolsTest_testCollision and ToolsTest_testBoundry are tests designed to check to make sure the collision detection system within the game is functioning. We used Junit test to test this as it was relatively simple to mock up occasions where collisions would occur and given its importance it allowed us to make sure that collisions didn't fail during implementation of the powerup class.

DifficultySelectionTest_testSelectEasy and it's partner tests DifficultySelectionTest_testSelectMedium, DifficultySelectionTest_testSelectHard and DifficultySelectionTest_testSelectVeryHard are a set of Junit tests that ensure that the settings of the difficulty levels are appropriately applied when a game starts. We tested this using Junit since it was easy to automate and would have been tricky to test manually as it would have required human comparison of many values which would lead to a high chance of human error. ManualTest_testMultipleDifficulties is a manual test that we designed to ensure that the game would show multiple difficulty levels on the difficulty select screen. To do this the tester would go through the game setup multiple times selecting a different difficulty each time. This had to be done manually as there is no easy way to test UI elements in LibGDX.

SaveTest_saveRaceTest is a large Junit test that checks to see if the save function of the game works as intended. We used a Junit test for this due to how time consuming it would be to test this manually (as it requires loading and saving every possible gamestate) and manual testing could have easily led to human error (as much of the data was hidden in the background of the program).

PrefsTest_integerTest, PrefsTest_floatTest, PrefsTest_arrayTest, PrefsTest_vector2Test and PrefsTest_boatTypeTest are a set of tests to ensure that the functional components of the save system load and save as intended. Junit was used for this as it was the most time efficient solution. PrefsTest_openThrowTest and PrefsTest_openNoThrowTest were a set of Junit tests that were used to check that the error handling system of the save system was working correctly. Junit was used as it saved time.

CollidableTest_testRockUpdate, CollidableTest_testBranchUpdate, CollidableTest_testLeafUpdate, CollidableTest_testInvulnUpdate, CollidableTest_testSpeedUpUpdate, CollidableTest_testLessDamageUpdate, CollidableTest_testLessTimeUpdate, CollidableTest_testHealUpdate are a set of tests that check to make sure that all the collidable objects moved on the screen correctly. For this test we used Junit as it was simple method testing. Also, due to the nature of the test we didn't use a large array of test values since the test only needed to determine if the objects moved in the correct direction.

CollidableTest_testRockDamage, CollidableTest_testBranchDamage, CollidableTest_testLeafDamage were all setup to ensure that the obstacles dealt the correct amount of damage for their type and by extension it tested the takeEffect() method within the Collidable class. Junit was used since this was simple method testing.

CollidableTest_testInvulnEffect, CollidableTest_testSpeedUpEffect, CollidableTest_testLessDamageEffect, CollidableTest_testLessTime, CollidableTest_testHealEffect, ManualTest_explanationTest were a set of tests that were designed to ensure that the powerups had the desired effects on the boats. The CollidableTest_testLessTime was designed slightly differently to the others in that multiple values were passed to it to ensure that the time was changed correctly in fringe cases (time = 0 for example). The other tests didn't require this as their effects were simpler. Junit test was used here as it allowed for more clarity when it came to the test results compared to say manual testing (most of the effects change game logic vs having an obvious visual effect).

ManualTest_explanationTest this is a manual test that is designed to make sure that the tutorial provided is adequate for the game. This was done by having the tester open the game and navigate to the help screen and then reporting on how complete they felt the instructions were. Manual testing had to be used here as it is impossible to quantify the results of this test mathematically.

As stated above, further data on all of the tests performed can be found in the Testing Documentation file which is linked here and also above.
https://team-13-rlc.github.io/pdfs/TestDoc2.pdf

c)

Evidence Links:
The Testing Documentation file and the Testability matrix can both be found elsewhere in the document.
Testing reports and coverage screenshots can be found here:
https://team-13-rlc.github.io/pdfs/TestEvidence2.pdf

Bibliography:

[1] Tom Grill, gdx-testing, GitHub [online]. Available:https://github.com/TomGrill/gdx-testing
[Accessed 3rd February 2021]